Correction Working with Methods and Encapsulation - 0

A - null:0

Methods can have same name as the class. Student() and Student(String, int) are methods and not constructors of the class, note the void return type of these methods.

As no constructors are provided in the Student class, java compiler adds default no-arg constructor. That is why the statement Student s = new Student(); doesn't give any compilation error.

Default values are assigned to instance variables, hence null is assigned to name and 0 is assigned to age.

Correction Working with Methods and Encapsulation - 1

D - Infinite loop

This is an example of pass-by-value scheme. x of checkAndIncrement method contains the copy of variable x defined in main method. So, changes done to x in checkAndIncrement method are not reflected in the variable x of main. x of main remains 1 as code inside main is not changing its value.

Correction Working with Methods and Encapsulation - 2

C - Compilation Error

In this case “if - else if” block is used and not “if - else” block.

90000 is assigned to variable ‘price’ but you can assign parameter value or call some method returning double value, such as:

‘double price = currentTemp();’.

In these cases compiler will not know the exact value until runtime, hence Java Compiler is not sure which boolean expression will be evaluated to true and so variable model may not be initialized.

Correction Working with Methods and Encapsulation - 3

B - Make ‘passportNo’ private and provide a public method getPassportNo() which will return its value.

‘passportNo’ should be read-only for any other class.

This means make ‘passportNo’ private and provide public getter method. Don’t provide public setter as then ‘passportNo’ will be read-write property.

Correction Working with Methods and Encapsulation - 4

D - Compilation error

main method's parameter variable name is "args" and it is a local to main method.

So, same name "args" can't be used directly within the curly brackets of main method.

Correction Working with Methods and Encapsulation - 5

A - Encapsulation

Correction Working with Methods and Encapsulation - 6

D -

```
Happy New Year!
Happy New Year!
```

It is pass-by-reference scheme.

Initially, msg = “Happy New Year!”

Call to method change(Message) doesn’t modify the msg property of passed object rather it creates another Message object and modifies the msg property of new object to “Happy Holidays!”

So, the instance of Message referred by obj remains unchanged.

Hence in the output, you get:

Happy New Year!

Correction Working with Methods and Encapsulation - 7

B - int version char version

Correction Working with Methods and Encapsulation - 8

A - 25-60

This question checks your knowledge of pass-by-value and pass-by-reference schemes.

In below statements: student

means student inside main method.

On execution of main method: student
–> {“James”, 25}, marks

= 25.

On execution of review method: stud –> {“James”, 25} (same object referred by student

), marks = 25 (this marks is different from the marks defined in main method). marks = 35 and stud.marks = 60. So at the end of review method: stud –> {“James”, 60}, marks = 35.

Correction Working with Methods and Encapsulation - 9

A -

```
private static String process(int [] arr, int start, int end) {
  return null;
}
```

It is clear from Line 5 that, method name should be process, it should be static method, it should accept 3 parameters (int[], int, int) and its return type must be String.

Correction Working with Methods and Encapsulation - 10

B -

```
public class Clock {
  private String model;
  public String getModel() { return model; }
  public void setModel(String val) { model = val; }
}
```

Encapsulation is all about having private instance variable and providing public getter and setter methods.

Correction Working with Methods and Encapsulation - 11

B - report is the method name

It is good practice to have first character of class name in upper case, but it is not mandatory.

student can be either class name or reference variable name.

Syntax to invoke static method is: Class_Name.method_name(); OR reference_variable_name.method_name();

Syntax to invoke instance method is: reference_variable_name.method_name();

If student represents class_name or refernce_variable_name, then report might be the static method of the class.

If student represents reference_variable_name, then report is the instance method of the class.

In both the cases, report must be the method name.

Type of argument cannot be found out by looking at above syntax.

Correction Working with Methods and Encapsulation - 12

D - double version: 20.0

int can be converted to double but Integer type can’t be converted to Double type as Integer and Double are siblings (both extends from Number class) so can’t be casted to each other.

add(10.0, new Integer(10)); => 1st parameter is tagged to double primitive type and 2nd parameter is converted to int, is tagged to double primitive type as well. So, add(double, double); method is invoked.

Correction Working with Methods and Encapsulation - 13

C -

```
Happy New Year!
Happy Holidays!
```

It is pass-by-reference scheme.

Initially, msg = “Happy New Year!”

Call to method change(Message) modifies msg property of passed object to “Happy Holidays!”

Hence in the output, you get:

```
Happy New Year!
Happy Holidays!
```

Correction Working with Methods and Encapsulation - 14

A - Compilation error

A constructor can call another constructor by using this(…) and not the constructor name.

Hence Student(“James”, 25); causes compilation error.

Correction Working with Methods and Encapsulation - 15

A - Point(19, 35, -1) Point(19, 40, 0)

Point class correctly overrides the toString() method. Even though variable x is static, but it can be easily accessed by instance method toString().

Variables x, y and z are declared with default scope, so can be accessed in same package. There is no compilation error in the code.

There is only one copy of static variable for all the instances of the class. Variable x is shared by p1 and p2 both.

p1.x = 17; sets the value of static variable x to 17, p2.x = 19; modifies the value of static variable x to 19. As there is just one copy of x, hence p1.x = 19

Please note: p1.x and p2.x don't cause any compilation error but as this syntax creates confusion, so it is not a good practice to access the static variables or static methods using reference variable, instead class name should be used. Point.x is the preferred syntax.

Each object has its own copy of instance variables, so just before executing Line n1, p1.y = 35 & p1.z = -1 AND p2.y = 40 & p2.z = 0

Output is:

Point(19, 35, -1)

Point(19, 40, 0)

Correction Working with Methods and Encapsulation - 16

B - null

public void Test() is method and not constructor, as return type is void.

method can have same name as the class name, so no issues with Test() method declaration.

As there are no constructors available for this class, java compiler adds following constructor.

public Test() {}

Test obj = new Test(); invokes the default constructor but it doesn't change the value of name property (by default null is assigned to name property)

System.out.println(obj.name); prints null.

Correction Working with Methods and Encapsulation - 17

D - this.color = color;

Instance variable color is shadowed by the parameter variable color of parameterized constructor. So, color = color will have no effect, because short hand notation within constructor body will always refer to LOCAL variable. To refer to instance variable, this reference is needed. Hence 'this.color = color;' is correct.

'color = GREEN;' and 'this.color = GREEN;' cause compilation error as GREEN is not within double quotes(“”).

NOTE: 'color = “GREEN”;' will only assign 'GREEN' to local variable and not instance variable but 'this.color = “GREEN”;' will assign 'GREEN' to instance variable.

Correction Working with Methods and Encapsulation - 18

B - Compilation Error

i and j cannot be declared private as i and j are local variables.

Only final modifier can be used with local variables.

Correction Working with Methods and Encapsulation - 19

B - If the class does not define any constructors explicitly.

Default constructor (which is no-argument constructor) is added by Java compiler, only if there are no constructors in the class.

Correction Working with Methods and Encapsulation - 20

A -

```
private static String[] process(int [] arr, int start, int end) {
  return null;
}
```

C -

```
private static String process(int [] arr, int start, int end) {
  return null;
}
```

D -

```
private static int[] process(int [] arr, int start, int end) {
  return null;
}
```

It is clear from Line 5 that, method name should be process, it should be static method, it should accept 3 parameters (int[], int, int).

As process(arr, 3, 8) is passed as an argument of System.out.println method, hence process method's return type can be anything apart from void. println method is overloaded to accept all primitive types, char [], String type and Object type. int[] are String [] are of Object type.

In the given options, method specifying int as return type cannot return null as null can't be assigned to primitive type. int process(...) would cause compilation error.

Correction Working with Methods and Encapsulation - 21

E - Compilation error

Variable y is private so it cannot be accessed outside the boundary of Point class.

p1.y and p2.y used inside Test class, cause the compilation error.

Correction Working with Methods and Encapsulation - 22

A -

```
Good Morning!
Good Evening!
```

Greetings g1 = new Greetings(); invokes no-arg constructor.

No-arg constructor calls parameterized constructor with the argument “Good Morning!”

Parameterized constructor assigns “Good Morning!” to msg variable of the object referred by g1.

Greetings g2 = new Greetings(“Good Evening!”); invokes parameterized constructor, which assigns “Good Evening!” to msg variable of the object referred by g2.

g1.display(); prints Good Morning!

g2.display(); prints Good Evening!

Correction Working with Methods and Encapsulation - 23

B - final static int x = 10;

Fields declared with final are constant fields.

Correction Working with Methods and Encapsulation - 24

C -

```
2
1
```

It is pass-by-value scheme. On method invocation, parameter variable num gets a copy and changes are made to this copy inside the method. Original value of i1 stay intact.

Correction Working with Methods and Encapsulation - 25

A - Compilation error

add(10.0, new Double(10.0)); is an ambiguous call as compiler can’t decide whether to convert 1st argument to Double reference type or 2nd argument to double primitive type.

So, add(10.0, new Double(10.0)); causes compilation error.

Correction Working with Methods and Encapsulation - 26

C - Line 8 causes compilation failure

Line 4 code shadows the variable at Line 2. msg variable created at Line 4 is a local variable and should be initialized before it is used.

Initialization code is inside if-block, so compiler is not sure about msg variable's initialization. Hence, Line 8 causes compilation failure.

Correction Working with Methods and Encapsulation - 27

D - public Planet(String str) {}

Constructor has the same name as the class, doesn’t have return type and can accept parameters.

Correction Working with Methods and Encapsulation - 28

B - Replace Student(“James”, 25); with this(“James”, 25);

D - Add below code in the Student class:

```
void Student(String name, int age) {
  this.name = name;
  this.age = age;
}
```

First find out the reason for compilation error, all the options are giving hint :)

no-arg constructor of Student class calling another overloaded constructor by the name and this causes compilation error. This problem can be fixed in 2 ways:

1st one: replace Student(“James”, 25); with this(“James”, 25) OR 2nd one: add void Student(String, int) method in the Student class.

Method can have same name as the class name and constructor can call other methods.

Correction Working with Methods and Encapsulation - 29

C -

```
null
null
```

Greetings g1 = new Greetings(); invokes no-arg constructor. Property msg (of object referred by g1) is assigned to null.

Greetings g2 = new Greetings("Good Evening!"); invokes parameterized constructor, which assigns "Good Evening!" to msg of object referred by g2.

g1.display(); prints null

Again we have same call g1.display(); which prints null.

NOTE: We haven't called display() on object referred by g2.

Correction Working with Methods and Encapsulation - 30

B - Replace add(10, 20); by add(null, null);

C - Remove add(int i, int j) method declaration and definition.

D - Replace add(10, 20); by add(new Integer(10), new Integer(20));

Method add is overloaded in Test class. Which overloaded method is invoked is decided at the compile time. add(10, 20); tags to int version as 10, 20 are int literals and direct match is available. So without any changes, above code prints “int version” on to the console.

To print “Integer version” on to the console, add(Integer, Integer); method needs to be invoked. Let’s check all the options one by one:

“Remove add(int i, int j) method declaration and definition.” add(10, 20); => auto-boxing will convert literal 10 and 20 to Integer instances and will call the add(Integer, Integer) method. So this option is valid.

Replace add(10, 20); by add(new Integer(10), new Integer(20)); => This statement is specifically calling add(Integer, Integer); So this option is also valid.

Replace add(10, 20); by add(10.0, 20.0); 10.0 and 20.0 are default literals and can’t be mapped to int or Integer types, hence this gives compilation error. Not a valid option.

Replace add(10, 20); by add(null, null); As Integer is reference type hence add(null, null); maps to add(Integer, Integer); So this is also a valid option.

Correction Working with Methods and Encapsulation - 31

A - static method

print() is static method of class Test. So correct syntax to call method print() is Test.print();

but static methods can also be invoked using reference variable: obj.print(); Warning is displayed in this case.

Even though obj has null value, we don’t get NullPointerException as objects are not needed to call static methods.

Correction Working with Methods and Encapsulation - 32

C - Compilation error

color is LOCAL variable and it must be initialized before it can be used.

As area is not compile time constant, java compiler doesn’t have an idea of the value of variable area.

There is no else block available as well.

So compiler cannot be sure of whether variable color will be initialized or not, therefore System.out.println(color); causes compilation error.

Correction Working with Methods and Encapsulation - 33

B - Replace add(10, 20); by add(null, null);

C - Remove add(int i, int j) method declaration and definition.

D - Replace add(10, 20); by add(new Integer(10), new Integer(20));

Method add is overloaded in Test class. Which overloaded method is invoked is decided at the compile time. add(10, 20); tags to int version as 10, 20 are int literals and direct match is available. So without any changes, above code prints “int version” on to the console.

To print “Integer version” on to the console, add(Integer, Integer); method needs to be invoked. Let’s check all the options one by one:

“Remove add(int i, int j) method declaration and definition.” add(10, 20); => auto-boxing will convert literal 10 and 20 to Integer instances and will call the add(Integer, Integer) method. So this option is valid.

Replace add(10, 20); by add(new Integer(10), new Integer(20)); => This statement is specifically calling add(Integer, Integer); So this option is also valid.

Replace add(10, 20); by add(10.0, 20.0); 10.0 and 20.0 are default literals and can’t be mapped to int or Integer types, hence this gives compilation error. Not a valid option.

Replace add(10, 20); by add(null, null); As Integer is reference type hence add(null, null); maps to add(Integer, Integer); So this is also a valid option.

Correction Working with Methods and Encapsulation - 34

A - static method

print() is static method of class Test. So correct syntax to call method print() is Test.print();

but static methods can also be invoked using reference variable: obj.print(); Warning is displayed in this case.

Even though obj has null value, we don’t get NullPointerException as objects are not needed to call static methods.

Correction Working with Methods and Encapsulation - 35

C - Compilation error

color is LOCAL variable and it must be initialized before it can be used.

As area is not compile time constant, java compiler doesn’t have an idea of the value of variable area.

There is no else block available as well.

So compiler cannot be sure of whether variable color will be initialized or not, therefore System.out.println(color); causes compilation error.

Correction Working with Methods and Encapsulation - 36

B - Compilation error in TestBook class

Variable ‘book’ in main(String[]) method of TestBook class cannot be declared private as it is a local variable. Hence, there is a compilation error in TestBook class.

Only final modifier can be used with local variables.

Correction Working with Methods and Encapsulation - 37

D - Add below 2 methods in Circle class:

```
public double getRadius() {
  return radius;
}

public void setRadius(double radius) {
  this.radius = radius;
}
```

Circle class needs to be well encapsulated, this means that instance variable radius must be declared with private access modifier and getter/setter methods must be public, so that value in radius variable can be read/changed by other classes.

Out of the given options, below option is correct:

Add below 2 methods in Circle class:

```
public double getRadius() {
    return radius;
}

public void setRadius(double radius) {
    this.radius = radius;
}
```

Correction Working with Methods and Encapsulation - 38

E - Compilation error

Both the methods of Report class have same signature(name and parameters match).

Having just different return types don’t overload the methods and therefore Java compiler complains about duplicate generateReport() methods in Report class.

Correction Working with Methods and Encapsulation - 39

C -

```
VIRUS
VIRUS
```

Threat class doesn't specify any constructor, hence Java compiler adds below default constructor:

Threat() {super();}

Line n1 creates an instance of Threat class and initializes instance variable 'name' to "VIRUS". Variable 'obj' refers to this instance.

Line n2 prints VIRUS on to the console.

Line n3 invokes evaluate(Threat) method, as it is a static method defined in AvoidThreats class, hence `evaluate(obj);` is the correct syntax to invoke it. Line n3 compiles successfully. On invocation parameter variable 't' copies the content of variable 'obj' (which stores the address to Threat instance created at Line n1). 't' also refers to the same instance referred by 'obj'.

On execution of Line n6, another Threat instance is created, its instance variable 'name' refers to "VIRUS" and 't' starts referring to this newly created instance of Threat class. Variable 'obj' of main(String[]) method still refers to the Threat instance created at Line n1. So, 'obj' and 't' now refer to different Threat instances.

Line n7, assigns "PHISHING" to the 'name' variable of the instance referred by 't'. evaluate(Threat) method finishes its execution and control goes back to main(String[]) method.

Line n4 is executed next, print() method is invoked on the 'obj' reference and as obj.msg still refers to "VIRUS", so this statement prints VIRUS on to the console.

Hence in the output, you get:

```
VIRUS
VIRUS
```

Correction Working with Methods and Encapsulation - 40

D - It executes successfully and prints 30 on to the console

i1 is a static variable and i2 is an instance variable. Preferred way to access static variable i1 inside add() method is by using ‘i1’ or ‘Test.i1’. Even though ‘this.i1’ is not the recommended way but it works.

And instance variable i2 can be accessed inside add() method by using ‘i2’ or ‘this.i2’. Hence, Line n1 compiles successfully.

As add() is an instance method of Test class, so an instance of Test class is needed to invoke the add() method. `new Test().add()` correctly invokes the add() method of Test class and returns 30. Line n2 prints 30 on to the console.

Correction Working with Methods and Encapsulation - 41

F - It prints 5 on to the console

As both the classes: Square and TestSquare are in the same file, hence variables ‘length’ and ‘sq’ can be accessed using dot operator. Given code compiles successfully.

Line n1 creates an instance of Square class and ‘sq1’ refers to it. sq1.length = 10 and sq1.sq = null.

Line n2 creates an instance of Square class and ‘sq2’ refers to it. sq2.length = 5 and sq2.sq = null.

On execution of Line n3, sq1.sq = sq2.

Line n4: System.out.println(sq1.sq.length); =>
System.out.println(sq2.length); => Prints 5 on to the console.

Correction Working with Methods and Encapsulation - 42

B - It compiles successfully and on execution prints 1233 on to the console

If choice is between implicit casting and variable arguments, then implicit casting takes precedence because variable arguments syntax was added in Java 5 version.

m('A'); is tagged to m(int) as 'A' is char literal and implicitly casted to int.

m('A', 'B'); is tagged to m(int, int) as 'A' and 'B' are char literals and implicitly casted to int.

m('A', 'B', 'C'); is tagged to m(char...)

m('A', 'B', 'C', 'D'); is tagged to m(char...)

There is no compilation error and on execution output is: 1233

Correction Working with Methods and Encapsulation - 43

A - Delete Line n1, Line n2 and Line n3

speed method is correctly overloaded in Car class as both the methods have different signature: speed(Byte) and speed(byte…). Please note that there is no rule regarding return type for overloaded methods, return type can be same or different.

`new Car().speed(b);` tags to speed(Byte) as boxing is preferred over variable arguments. Code as is prints DARK on to the console.

Variable arguments syntax '…' can be used only for method parameters and not for variable type and type-casting. Hence the option of replacing Line n4 and Line n5 are not correct.

If you delete speed(Byte) method, i.e. Line n1, Line n2 and Line n3, then `new Car().speed(b);` would tag to speed(byte…) method and on execution would print LIGHT on to the console.

Correction Working with Methods and Encapsulation - 44

C - private, public

‘private’ is most restrictive, then comes ‘default (with no access modifier specified)’, after that ‘protected’ and finally ‘public’ is least restrictive.

Correction Working with Methods and Encapsulation - 45

A - Line n3 causes compilation error

E - Line n7 causes compilation error

H - Line n5 causes compilation error

i1 is an instance variable and i2 is a static variable.

Instance method can access both instance and static members. Hence, Line n1 and Line n2 compile successfully.

Static method can access only static members. Hence, Line n3 [accessing instance variable i1], Line n5 [accessing instance method change1(int)] and Line n7 [accessing instance variable i1] cause compilation error.

Correction Working with Methods and Encapsulation - 46

D - Compilation error

At Line n2, local variable ‘str’ shadows the static variable ‘str’ created at Line n1. Hence, for the expression `str + "SIMPLE"`, Java compiler complains as local variable ‘str’ is not initialized.

Correction Working with Methods and Encapsulation - 47

D - All six statements

`private void emp() {}` is a valid method declaration.

Class name and method name can be same and that is why given method can be declared in any of the given classes: ‘emp’, ‘Emp’, ‘employee’, ‘Employee’, ‘Student’ and ‘*emp*’.

‘*emp*’ is also a valid Java identifier.

Correction Working with Methods and Encapsulation - 48

A -

```
LET IT GO!
NEVER LOOK BACK!
```

Message class doesn't specify any constructor, hence Java compiler adds below default constructor:

Message() {super();}

Line n1 creates an instance of Message class and initializes instance variable 'msg' to "LET IT GO!". Variable 'obj' refers to this instance.

Line n2 prints LET IT GO! on to the console.

Line n3 invokes change(Message) method, as it is a static method defined in TestMessage class, hence `change(obj);` is the correct syntax to invoke it. Line n3 compiles successfully. On invocation parameter variable 'm' copies the content of variable 'obj' (which stores the address to Message instance created at Line n1). 'm' also refers to the same instance referred by 'obj'.

Line n6, assigns "NEVER LOOK BACK!" to the 'msg' variable of the instance referred by 'm'. As 'obj' and 'm' refer to the same instance, hence obj.msg also refers to "NEVER LOOK BACK!". change(Message) method finishes its execution and control goes back to main(String[]) method.

Line n4 is executed next, print() method is invoked on the 'obj' reference and as obj.msg refers to "NEVER LOOK BACK!", so this statement prints NEVER LOOK BACK! on to the console.

Hence in the output, you get:

```
LET IT GO!
NEVER LOOK BACK!
```

Correction Working with Methods and Encapsulation - 49

C - 15:5 is printed on to the console

Each instance of the class contains separate copies of instance variable and share one copy of static variable.

There are 3 instances of Counter class created by main method and these are referred by ctr1, ctr2 and ctr3.

As ‘ctr’ is a static variable of Counter class, hence ctr1.ctr, ctr2.ctr and ctr3.ctr refer to the same variable. In fact, ‘Counter.ctr’ is the preferred way to refer the static variable ‘ctr’ but ctr1.ctr, ctr2.ctr and ctr3.ctr are also allowed.

As ‘count’ is an instance variable, so there are 3 separate copies: ctr1.count, ctr2.count, ctr3.count.

On the completion of for loop: ctr1.count = 5, ctr2.count = 5 and ctr3.count = 5 and Counter.ctr = 15.

15:5 is printed on to the console.

Correction Working with Methods and Encapsulation - 50

E - 120

calculate method is correctly overloaded as both the methods have different signature: calculate(int, int) and calculate(byte, byte). Please note that there is no rule regarding return type for overloaded methods, return type can be same or different.

`new Calculator().calculate(b, i)` tags to `calculate(int, int)` as byte value is implicitly casted to int type.

Given code compiles successfully and on execution prints 120 on to the console.

Correction Working with Methods and Encapsulation - 51

D - No changes are necessary

As member variables ‘testNo’ and ‘desc’ are declared with no explicit access specifier, this means these variables have package scope, hence these variables are accessible only to classes within the same package. Hence, no changes are necessary.

If you use private, then instance variables will not be accessible to any other classes, even within the same package.

If you use protected, then instance variables will be accessible to the subclasses outside ‘com.udayankhattry.oca’ package.

If you use public, then instance variables will be accessible to all the classes.